СУ „Св. Климент Охридски“, ФМИ

Специалност „Софтуерно Инженерство“

Обектно-ориентирано програмиране, 2020-2021 г.

Задача за домашно № 3

Срок: 09.05.2021 г. 23:59

Важна информация

Инструкции

1. *Позволено е използването на всички библиотеки от STL*
2. *Не променяйте предоставяните публични интерфейси (методи и полета) на класовете, тъй като тези методи ще се използват в автоматични тестове и ако имат променена сигнатура тестовете няма да компилират и ще получите 0 точки. От вас се очаква да имплементирате дадените методи.*
3. *За да компилира кодът ви трябва всички методи да имат имплементация, дори да връщат грешен отговор.*
4. *Позволено е да добавяте други методи/класове, за да реализирате задачата. Тях няма да ги тестваме.*
5. *Не е позволено използването на външни библиотеки (които не са част от STL)*

Оценяване на домашното

- Част от точките за това домашно ще бъдат давани след покриването на автоматични тестове - за **коректно реализирана функционалност**
- За да получите тези точки, предадените от вас решения трябва да отговарят на следните критерии
    - Да съдържат указаните методи и имена на класове (ще ви бъде даден шаблон, върху който да работите) - позволено е да добавяте нови методи и класове, но **не е позволено** да променяте даденото от нас.
    - Предавайте единствено файлове съдържащи код - архиви съдържащи `.sln` файлове или каквито и да е други файлове, които не са `.cpp` или `.hpp` ще получават 0 точки на автоматичните тестове.
    - **Не предавайте** архиви от тип `.rar` - **ще се приемат архиви от тип `.zip`.** При получен архив от тип `.rar` (или друг тип, които не

може да бъде разархивиран от системата за тестване), отново получавате 0 точки на автоматичните тестове.
- Именувайте архива си по следния начин - `SI_R_HW3_<курс>_<група>_<факултетен номер>`. Архиви, които не спазват тази конвенция ще получат 0 точки на автоматичните тестове. (Пример: `SI_R_HW3_1_1_12345.zip`)
- Не променяйте имената на файлове, които получавате
- След разархивиране на архива, трябва да се получат 2 папки, с имена ‘1’ и ‘2’
- Може да тествате архивите си тук: [линк]
- Спазвайте следната структура на архива:

```
SI_R_HW3_<курс>_<група>_<факултетен номер>.zip
├── 1
│   ├── Counter.hpp
│   ├── Counter.cpp
│   ├── LimitedCounter.hpp
│   ├── LimitedCounter.cpp
│   ├── LimitedTwowayCounter.hpp
│   ├── LimitedTwowayCounter.cpp
│   ├── Semaphore.hpp
│   ├── Semaphore.cpp
│   ├── TwowayCounter.cpp
│   └── TwowayCounter.hpp
├── 2
│   ├── Developer.hpp
│   ├── Developer.cpp
│   ├── LeavingRequest.hpp
│   ├── LeavingRequest.cpp
│   ├── PromotionRequest.hpp
│   ├── PromotionRequest.cpp
│   ├── Request.hpp
│   ├── Request.cpp
│   ├── TeamLead.cpp
│   └── TeamLead.hpp
```

- Ако решението на някоя задача ви не се компилира, получавате 0 точки на автоматичните тестове за съответната задача
- Спазвайте практиките за обектно-ориентирано програмиране, коментирани на упражнения и лекции.

Задача 1 (4+1 точки - 3 точки от автоматични тестове)

Условие

Съществуват различни видове броячи - някои могат само да увеличават бройката, която пазят, други могат и да я намалят, а трети имат ограничение до колко могат да отброяват.

Клас `Counter`

Най-простият брояч - само нагоре, без ограничение.

- Конструктор без параметри: началната стойност е 0 и стъпката на брояча е 1
- Конструктор с 1 параметър `int initial`: началната стойност е `initial`, а стъпката е 1
- Конструктор с 2 параметъра `int initial, unsigned step`: началната стойност е `initial`, а стъпката е `step`
- `increment()`: увеличава текущата стойност със стъпката на брояча
- `getTotal()`: връща `int` - текущата отброена стойност
- `getStep()`: връща `unsigned` - стъпката на брояча (не трябва да може да бъде променяна)

Клас `TwowayCounter`

Брояч, който може и да намалява отброяваната стойност.

Освен всичко изброено в `Counter`, съдържа и:

- `decrement()`: намалява текущата стойност със стъпката на брояча

Клас `LimitedCounter`

Брояч, който отброява само до дадена максимална стойност.

- Конструктор с 1 параметър `int max`: максималната стойност е `max`, началната е 0, а стъпката е 1
- Конструктор с 2 параметъра `int max, int initial`: максималната стойност е `max`, началната е `initial`, а стъпката е 1

- Конструктор с 3 параметъра `int max, int initial, unsigned step`: максималната стойност е `max`, началната е `initial`, а стъпката е `step`
- `increment()`: увеличава текущата стойност със стъпката на брояча само ако няма да надмине максималната
- `getMax()`: връща `int` - максималната стойност на брояча
- `getTotal()`: същия като този на `Counter`
- `getStep()`: същия като този на `Counter`

Клас `LimitedTwowayCounter`

Той е и LimitedCounter и TwowayCounter едновременно: може да отброява нагоре до определена максимална стойност и надолу до определена минимална стойност.

- Конструктор с 2 параметъра `int min, int max`: минималната стойност е `min`, максималната стойност е `max`, началната е 0, а стъпката е 1
- Конструктор с 3 параметъра `int min, int max, int initial`: минималната стойност е `min`, максималната стойност е `max`, началната е `initial`, а стъпката е 1
- Конструктор с 4 параметъра `int min, int max, int initial, unsigned step`: минималната стойност е `min`, максималната стойност е `max`, началната е `initial`, а стъпката е `step`
- `increment()`: същия като на `LimitedCounter`
- `decrement()`: намаля текущата стойност със стъпката на брояча само ако няма да стане по-ниска от минималната
- `getMin()`: връща минималната стойност на брояча
- `getMax()`: същия като този на `LimitedCounter`
- `getTotal()`: същия като този на `Counter`
- `getStep()`: същия като този на `Counter`

Клас `Semaphore` (бонус)

Най-простия бинарен семафор - това е `LimitedTwowayCounter`, който има минимална стойност 0, максимална стойност 1 и стъпка 1. Използва се от процесите в операционните системи за синхронизационни цели. (Повече информация - след 1 година в курса по ОС)

- Конструктор без параметри - началната стойност на брояча е 0.
- Конструктор с един параметър `bool` - при `true` началната стойност на брояча е 1, а при `false` е 0.
- `isAvailable()`: връща `bool`, показващ дали стойността на брояча е над 0
- `wait()` - прави същото като `decrement()` на `LimitedTwowayCounter`

- `signal()` - прави същото като `increment()` на `LimitedTwowayCounter`

Задача 2: (6 точки - 4.5 точки от автоматични тестове)

Условие

Направете система, която да дава възможност за комуникация чрез заявки между разработчици и техния ръководител на екип. Имплементирайте функционалности за изпращане на заявка за напускане и заявка за повишаване на заплатата.

Клас `Developer`

Разработчик.

- Име (низ);
- Заплата (double);
- Ръководител на екип (указател към обект от клас `TeamLead`);
- Конструктор с 1 параметър `const string& name`: името е `name`, заплатата е `0`, указателят сочи към `nullptr`;
- `getName()`: връща `string` - текущата стойност на името;
- `getSalary()`: връща `double` - текущата стойност на заплатата;
- `getTeamLead()`: връща `TeamLead *` - текущата стойност на указателя към ръководителя;
- `setInitialSalary(double amount)`: присвоява за заплатата подадената сума `amount`, само ако нейната текуща стойност е `0`; (Ако не е да не присвоява стойност)
- `void sendLeavingRequest();`: подава към ръководителя `LeavingRequest` с името на разработчика;
- `void sendPromotionRequest(double amount);`: подава към ръководителя `PromotionRequest` с името на разработчика и количеството на заплатата, подадено в аргумента на функцията;

Клас `TeamLead`

Ръководител на екип.

Освен всичко изброено в `Developer`, съхранява и:

- вектор с указатели към обекти от клас `Developer`; (ръководителят не е част от екипа)
- заявки за напускане (`LeavingRequest`);
- заявки за повишаване на заплатата (`PromotionRequest`);
- Конструктор с 2 параметъра `const string& name, double salary`: името е `name`, заплатата е `salary`, а указателят към ръководителят на екипа сочи към текущият обект;
- `getTeam()`: връща `vector<Developer *>` - текущият вектор, който представлява екипа;
- `void addDeveloperToTeam(Developer * developer, double salary);`: добавя разработчик към екипа и задава заплатата му да бъде със стойност `salary`. Трябва за разработчикът да се промени, че вече има ръководител на екипа; (Възможно е да има разработчици в екипа с еднакви имена. Съобразете дали `developer` не сочи към `nullptr`.)
- `void removeDeveloperFromTeam(const string& name);`: премахва разработчик от екипа по подадено име `name`. (Ако не се намери търсения разработчик, не последва действие. Ако намери разработчици с еднакво име премахва последния.)
- `void increaseTeamSalariesBy(double amount);`: повишава заплатите на разработчиците в екипа със стойност `amount`;
- `void decreaseTeamSalariesBy(double amount);`: понижава заплатите на разработчиците в екипа със стойност `amount`;
- `void addLeavingRequest(const LeavingRequest& leavingRequest);`: добавя на съхранение заявка за напускане;
- `void addPromotionRequest(const PromotionRequest& promotionRequest);`: добавя на съхранение заявка за повишаване на заплатата;
- `void fulfillLeavingRequests();`: изпълнява всички съхранени заявки за напускане, като премахва от екипа изпратилите заявките, променя техният ръководител (`TeamLead *` да сочи към `nullptr`) и изчиства всички изпълнени заявки;
- `void fulfillPromotionRequests();`: изпълнява всички съхранени заявки за повишаване на заплатата, повишава заплатата на изпратилите заявките и изчиства всички изпълнени заявки;

Клас `Request`

Заявка.

- Съобщение (низ)
- Изпращач (низ) (името на този, който я изпраща)
- Брояч (цяло число) - започва от 0 и се увеличава с всеки създаден обект от тип `Request` или негови наследници;

- ID (цяло число) - уникален идентификатор, който има стойността на брояча в момента на създаването му (ако сме създали два обекта от тип `Request` или негови наследници, първият ще има ID със стойност 1, а вторият ID със стойност 2);
- Конструктор с 2 параметъра `const string& message, const string& sender`: съобщението е `message`, изпращачът е `sender`, броячът се увеличава с 1, ID приема стойността на брояча;
- `getMessage()` : връща `string` - текущата стойност на съобщението;
- `getSender()`: връща `string` - текущата стойност на изпращача;
- `getCount()`: връща `int` - текущата стойност на брояча;
- `getID()`: връща `int` - текущата стойност на ID;

Клас `LeavingRequest`

Заявка за напускане.

Освен всичко изброено в `Request`:

- Конструктор с 1 параметъра `const string& sender`: съобщението е `"I want to leave!"`, изпращачът е `sender`, броячът се увеличава с 1, ID приема стойността на брояча;

Клас `PromotionRequest`

Заявка за повишаване на заплатата.

Освен всичко изброено в `Request`:

- Количество (double) - количество, с което да се увеличи заплатата;
- Конструктор с 2 параметъра `const string& sender, double amount`: съобщението е `"I want a raise!"`, изпращачът е `sender`, количеството е `amount` , броячът се увеличава с 1, ID приема стойността на брояча;
- `getAmount()`: връща `double` - текущата стойност на количеството;

При решението на задачата и работатата с указатели НЕ трябва да се създават копия на обектите, сочени от указателите.